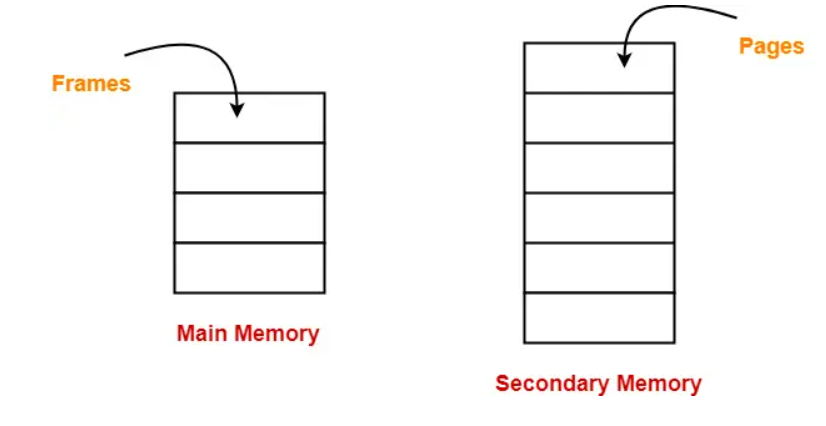
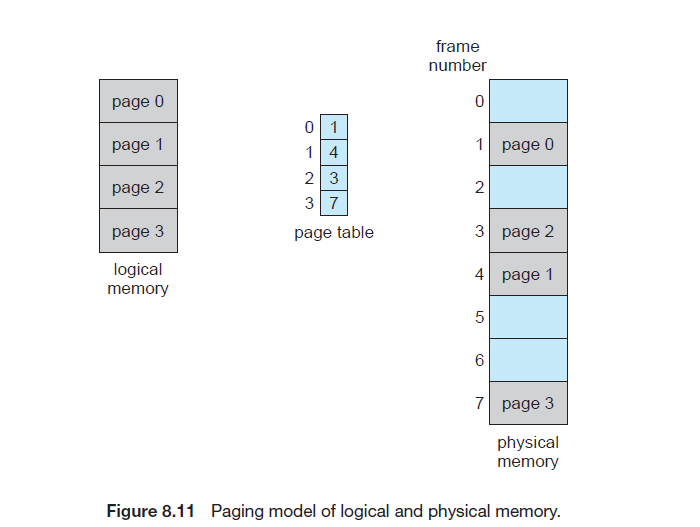
PAGING

Paging is a memory management technique and also Paging is a fixed size partitioning scheme in which process is divided into equal fixed size partitions. The partitions of secondary memory are called as **pages**. Physical memory is divided into same equal size partitions. The partitions of main memory are called as **frames**. Page size and frame size must be same. Pages are scattered in frames.



When a process is to be executed, its pages are loaded into any available memory frames from the backing store. Operating System will keep track of pages which are kept in main memory. For that purpose, it creates a one table called as page table. OS maintains a page table for each process. Page table is also stored on physical memory frames.

Entries of page table of each process = total no of pages of that process



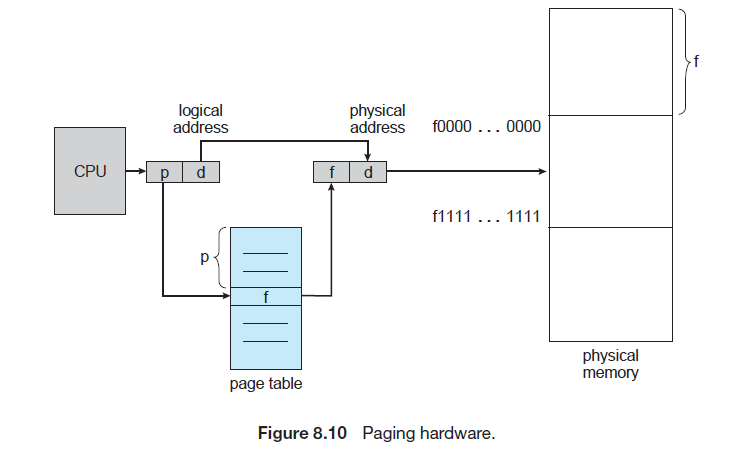
**Basic Method:**

The CPU always generates a logical address. In order to access the main memory always a physical address is needed.

The**logical address generated by CPU always consists of two parts:**

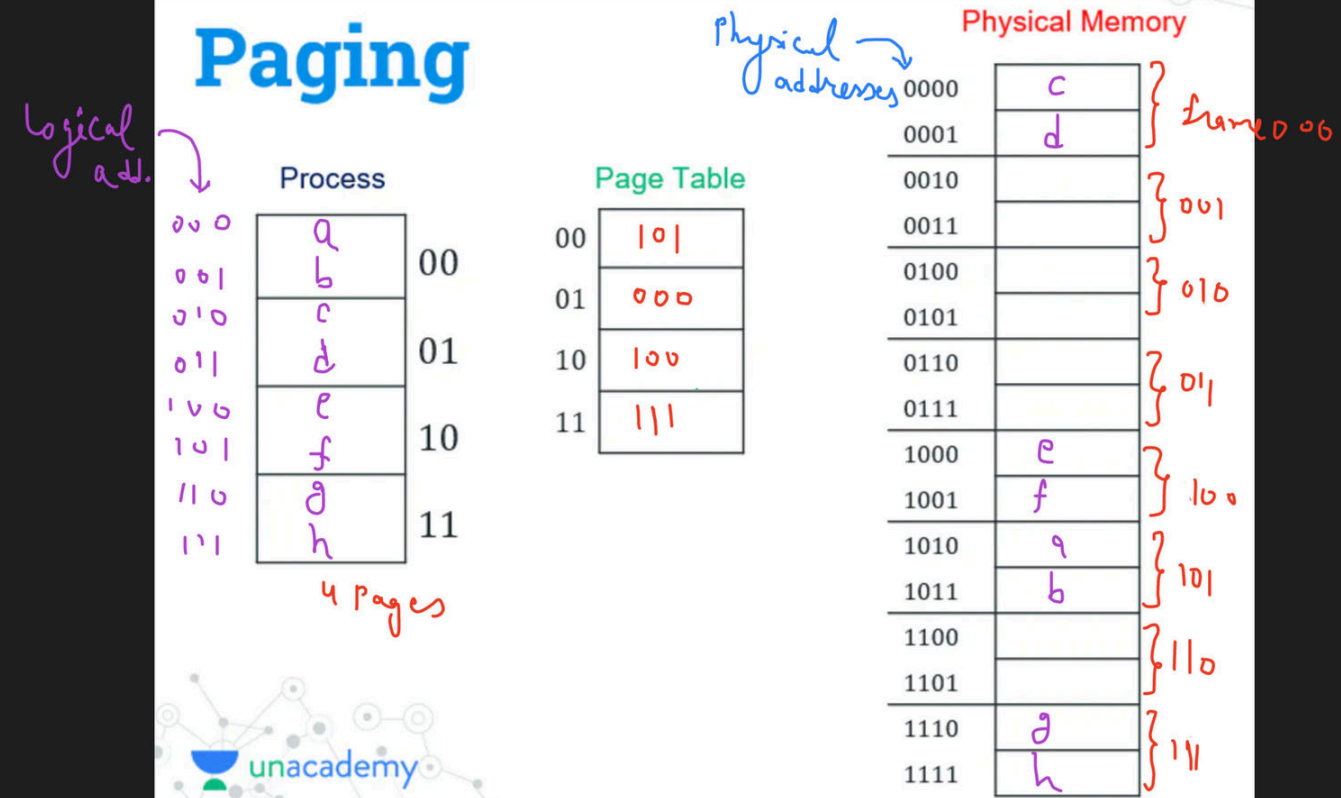
1. Page Number(p)
2. Page Offset (d)

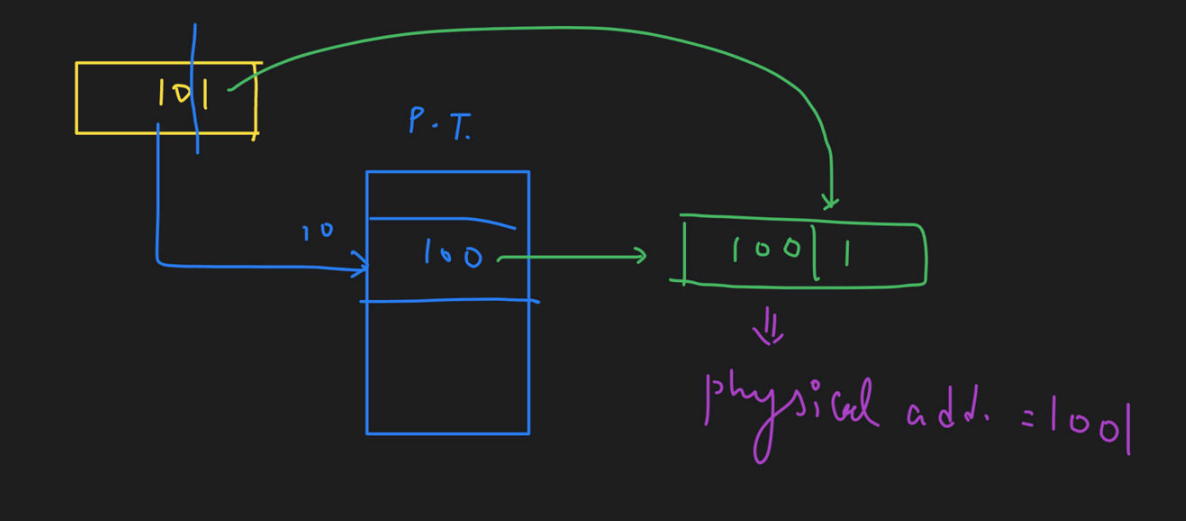
where, **Page Number** is used to specify the specific page of the process from which the CPU wants to read the data. and it is also used as an index to the page table and **Page offset** is mainly used to specify the specific word on the page that the CPU wants to read. The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.



**Example:**

CPU generates logical address 101.that means p=10 and d=1. Indexing into the page table, we find that page 10 is in frame 100. And frame 100 added to displacement 1 we get physical address 1001.

****



**Hardware Support:**

The use of registers for the page table is satisfactory if the page table is reasonably small (for example, 256 entries). Most contemporary computers, however, allow the page table to be very large (for example, 1 million entries). For these machines, the use of fast registers to implement the page table is not feasible. Rather, the page table is kept in main memory, and a **page-table base register (PTBR)** points to the page table. Changing page tables requires changing only this one register, substantially reducing context-switch time.

But there is a problem with this approach and that is with the time required to access a user memory location. Suppose if we want to find the location i, we must first find the index into the page table by using the value in the PTBR offset by the page number for I. And this task requires memory access. It then provides us the frame number which is combined with the page offset in order to produce the actual address. After that, we can then access the desired place in the memory.

With the above scheme, two memory accesses are needed in order to access a byte (one for the page-table entry and one for byte). Thus, memory access is slower by a factor of 2 and in most cases, this scheme slowed by a factor of 2.

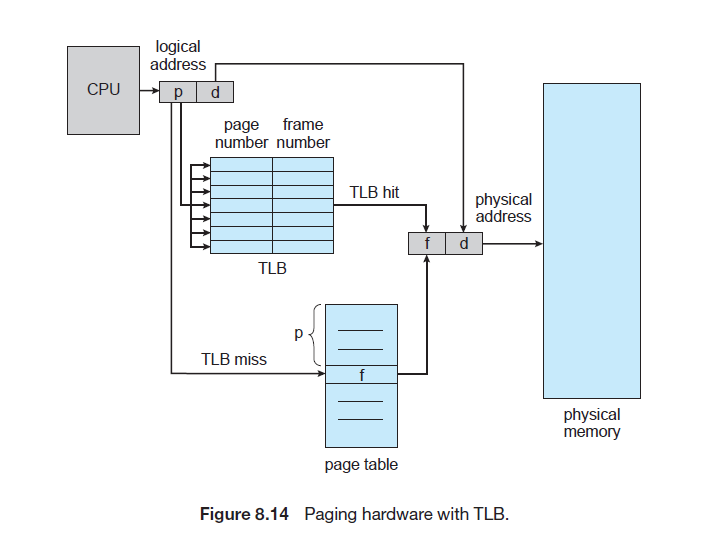
There is the standard solution for the above problem that is to use a special, small, and fast-lookup hardware cache that is commonly known as Translation look-aside buffer(TLB).

* TLB is associative and high-speed memory.
* Each entry in the TLB mainly consists of two parts: a key(that is the tag) and a value.
* When associative memory is presented with an item, then the item is compared with all keys simultaneously. In case if the item is found then the corresponding value is returned.
* The search with TLB is fast though the hardware is expensive.
* The number of entries in the TLB is small and generally lies in between 32 and 1024.

**TLB** is used with **Page Tables** in the following ways:

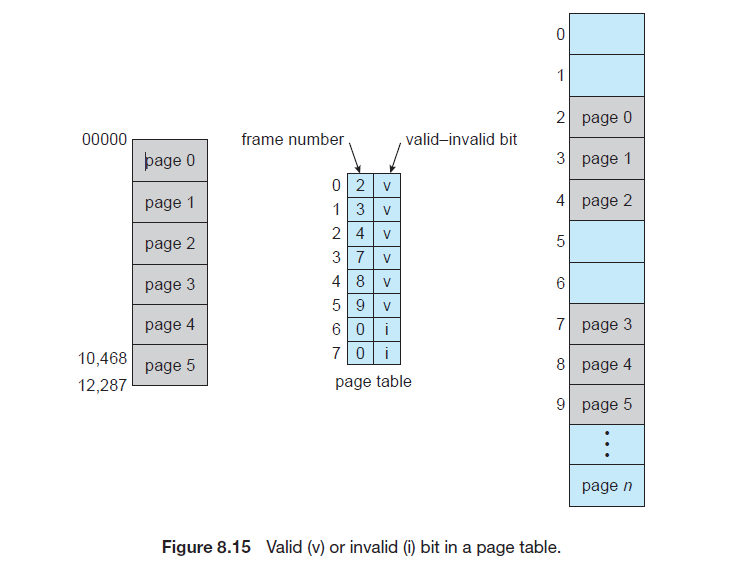
The TLB contains only a few of the page-table entries. Whenever the logical address is generated by the CPU then its page number is presented to the TLB.

* If the page number is found, then its frame number is immediately available and is used in order to access the memory. The above whole task may take less than 10 percent longer than would if an unmapped memory reference were used.
* In case if the page number is not in the TLB (which is known as**TLB miss**), then a memory reference to the Page table must be made.
* When the frame number is obtained it can be used to access the memory. Additionally, page number and frame number is added to the TLB so that they will be found quickly on the next reference.
* In case if the **TLB is already full of entries** then the Operating system must select o**ne for replacement.**



**Protection:**

Memory protection in a paged environment is accomplished by protection bits associated with each frame. Normally, these bits are kept in the page table. One additional bit is generally attached to each entry in the page table: a valid–invalid bit. When this bit is set to valid, the associated page is in the process’s logical address space and is thus a legal (or valid) page. When the bit is set to invalid, the page is not in the process’s logical address space. Illegal addresses are trapped by use of the valid–invalid bit. The operating system sets this bit for each page to allow or disallow access to the page.

****

Addresses in pages 0, 1, 2, 3, 4, and 5 are mapped normally through the page table. Any attempt to generate an address in pages 6 or 7, however, will find that the valid–invalid bit is set to invalid, and the computer will trap to the operating system (invalid page reference).